"Douglas W. Jones" <jones@cs.uiowa.edu> on 02/10/2002 12:55:20 PM

To: vss@FEC
cc:

Subject: Late comment on Voting System Standards

After attending the west02 conference, I have an addtional suggestion.
My suggestion is available in HTML (more readable) in

http://www.cs.uiowa.edu/~jones/voting/fec2.html

and it is appended in plain text format below:

- -

Added Comments on the FEC's Voting System Standards

Douglas W. Jones
Department of Computer Science
MacLean Hall
University of Iowa
Iowa City, Iowa 52242

(319)335-0740
jones@cs.uiowa.edu

Submitted to the FEC by E-mail, Feb 10, 2002; This material is also indexed on the web at http://www.cs.uiowa.edu/~jones/voting/

After reading and commenting on the voting system standards, found at

http://fecweb1.fec.gov/pages/vss/vss.html

and after attending the Workshop on Election Standards and Technology in Washington on January 31 and February 1, I've had a chance to step back and think about some of the big-picture elements of the problem of setting appropriate voting system standards.

The Problem with Software Standards

In my first comments on the July 13 draft, I didn't say much about the coding standards. If memory serves me correctly, they were stated relatively briefly, and they were sufficiently subjective that I could live with them. In the Dec.?13 draft, however, things seem tighter, more objective, and therefore more problematic. Perhaps I just read them in more detail this time, but the net result was that I found the standards to be far more offensive.

As a general rule, coding guidelines are good, but objectively testable coding standards are bad. It is good to demand clear and readable code, but attempts to codify readability are rarely successful. The analogy with readability rules for the English language is useful. In the world of education, there have been many attempts to codify readability, particularly with regard to children's literature, but the very best of children's literature generally ignores these readability rules!

There are a few exceptions. My favorite, Dr. Seuss's Green Eggs and Ham, was apparently written as something of an exercise in adhering to an extremely strict vocabulary limit, and it's an extraordiary work of literature. Despite such exceptions, most children's literature written to meet such standards is mediocre and much of it is awful.

Furthermore, codifying specific stylistic programming standards prevents innovation on the part of system developers.

Therefore, I strongly urge that the objective coding standards found in Sections 4.2.3 and 4.2.4 of the voting system standard be dropped! What should replace this material? The answer is found in the section of the draft standard governing version control. The draft standard does not mandate the use of any specific version control methodology; no specific automated approach is required, and in fact, there is no requirement for automation.

Instead, the standard merely requires that an effective version control methodology be used! I propose a very similar requirement with regard to coding standards, with one addition. I propose that every voting system vendor be required to provide, as one of the deliverables required for the software audit process, a copy of the coding guidelines they have used in developing their software. Here is my proposal:

4 Software Standards
4.1 Scope
4.2 Software Design and Coding Standards
4.2.3 Software Development Guidelines

Software purpose-written for voting systems should be modular, but the exact nature of a software module depends on the particular programming language being used. Modules are not necessarily the same thing as source files; in some languages, it is logical to include multiple related modules in one file. Modules are not necessarily functions; in some languages, it is desirable to include multiple related functions in a module.

The developer of voting software is responsible for adopting clear guidelines for code modularity, where these guidelines make specific provisions for the programming language, development environment and operating system being used. These guidelines must:

a) Incorporate widely accepted practices followed by programmers using the language in question. Eccentric approaches to modularization are discouraged unless they have clearly documented benefits.

b) Clearly define how modules (logical units of a program) relate to source files (physical units of the program).

c) Ensure that the relationships between the modules making up the program are clearly documented. Which modules does each module depend on, and what is the nature of the dependency?

d) Ensure that the interface each module offers to other modules is clearly documented. What kinds of things does each module make available for use by other modules, and what must the users of these things know in order to make use of them. Variables, types and functions are among the kinds of things that a module may export and that must be documented.

e) Encourage the use of appropriate granularity of modularization. Excessively large modules are hard to read, and the relationships between large numbers of excessively small modules are hard to understand.

f) Ensure that each module clearly defined name, and so that these names are used in a uniform way.

g) Ensure that each identifier used in the program is named in a way that ensures readability. Mnemonic or self-explanatory names should be encouraged, as should regular naming conventions that allow the meanings of short identifiers to be immediately understood; widely used examples of the latter are that x and y, when used alone, are display coordinates, i, when used alone, is an array index, and p, when used alone, is a pointer.

h) Discourage use of obscure or difficult to understand features of the programming language, and where such features are used, require clear documentation that warns the reader of the obscurity, documents the function and justifies its use. Examples of such problematic features abound in C and C++; the (a?b:c) construct, the comma operator, confusion between = and == in Boolean expressions, and similar features should be singled out, but there are equally dangerous features in other languages.

i) Require typography that is not dependant on unusual display or editing environments. Preferred practice should allow the software to be printed and easily readable on standard 8.5 by 11 inch paper, and if this is not possible, it must be possible to print the software on 11 by 17 inch paper. This leads naturally to limits of around 80 to 160 characters per line of text.

4.2.4 Conformance to Development Guidelines

In general, software developed specifically for the voting application should conform to the coding guidelines required by Section 4.2.3. However, the purpose of these guidelines is to encourage the development of code that is relatively easy for a knowledgable programmer to read, and in general, rigid enforcement of objective guidelines has not been shown to be an effective way to meet this goal.

Therefore, departure from previously adopted guidelines may be permitted when, in the judgement of the programmer, the guidelines lead to less readable code. Source code reviewers may, however, question such deviation if the benefits of nonconformance are not obvious and are not documented.